ISSN 1746-7659, England, UK
Journal of Information and Computing Science
Vol. 8, No. 4, 2013, *pp*. 264-274

A Theoretical Study of Extreme Parallelism in Sequence Alignments

Chung-E Wang

Department of Computer Science, California State University, Sacramento

(*Received March27, 2013, accepted August 25, 2013*)

Abstract. In this paper, we describe fully parallelized architectures for one-to-one, one-to-many, and many-to-many sequence alignments using Smith-Waterman algorithm. The architectures utilize the principles of parallelism and pipelining to the greatest extent in order to take advantage of both intra-sequence and inter-sequence parallelization and to achieve high speed and throughput. First, we describe a parallelized Smith-Waterman algorithm for general single instruction, multiple data (SIMD) computers. The algorithm has an execution time of $O(m+n)$, where m and n are the lengths of the two biological sequences to be aligned. Next, we propose a very-large-scale integration (VLSI) implementation of the parallel algorithm. Thirdly, we incorporate a pipelined architecture into the proposed VLSI circuit, producing a pipelined processor that can align a query sequence with a database of sequences at the speed of $O(m+n+L)$, where m is the length of the query sequence and n and L are the maximum length and the number of sequences in the database, respectively. Finally, we make use of our pipeline architecture to perform all possible pairs of pair-wise alignments for a group of L sequences with a maximum sequence length of m in $O(mL)$ time. Checking all pairs of pair-wise alignments is essential to the overlap-layout-consensus (OLC) approach for de novo assembly.

Keywords: Sequence alignment, sequence database search, Smith-Waterman algorithm, parallel algorithm, VLSI circuit, pipelined architecture, de novo assembly, overlap-layout-consensus approach.

1. Introduction

A sequence alignment is a way of matching two biological sequences to identify regions of similarity that may indicate functional, structural or evolutionary relationships between the two sequences. Sequence alignment is a fundamental operation of many bioinformatics applications such as sequence assembly, sequence database search, and short read mapping. There are two different types of sequence alignments: global alignments and local alignments. Global alignment is to find the best alignment across the entire two sequences. Local alignment is to find regions of high similarity in parts of the sequences. Today, local alignments are often preferable. In this paper, we focus on local sequence alignments.

The Smith-Waterman algorithm [1] is the most sensitive algorithm for performing sequence alignment. Here sensitivity refers to the ability to find the optimal alignment. The Smith-Waterman algorithm uses a linear gap function which was later improved by Gotoh with affine gap penalties [2]. The Smith-Waterman algorithm requires $O(mn)$ computational steps, where m and n are the lengths of the two sequences to be aligned. $O(mn)$ is quite efficient for comparing a pair of sequences. However, it's inadequate for performing multiple alignments, a common task for sequence database searches. A sequence database search is to compare a query sequence with a database of sequences and identify database sequences that resemble the query sequence above a certain threshold. To search a database of L sequences, L sequence alignments must be performed. Thus, without any parallelization, a time of $O(mnL)$ is required to search a sequence database, where m is the length of the query sequence and n and L are the maximum sequence length and the number of sequences in the database, respectively.

Due to substantial improvements in multiprocessing systems and the rise of multi-core processors, parallel architectures such as Field Programmable Gate Arrays (FPGAs), Graphics Processing Units (GPUs) and Very-Large-Scale-Integration (VLSI) circuits have been used to accelerate the Smith-Waterman algorithm and sequence database searches [3-28]. However, most of these previously established enhancements can only speed up the algorithm by a constant factor. That is, most of these enhancements still require an execution time of $O(mn)$ to align two biological sequences and thus require $O(mnL)$ time for a sequence database search.

In [6], Hurley claimed without giving details that using *O(n)* processing elements a sequence database search can be done in *O(n+N)* time, where *n* is length of the query sequence and *N* is the size of the database. This is theoretically impossible. In theory, using *p* processing elements, you can only speedup an algorithm's execution time from *T* to *T/p*. Sequentially, the best execution time for a sequence database search is $O(n^2N)$, where *N* is the size of the database and *n* is the length of the query sequence and database sequences. Thus, the best can be achieved with *O(n)* processing elements is $O(n^2N)/O(n)$ which is *O(nN)* not *O(n+N)*.

In [12], Rajko et al. showed that they can use *p* processors to cut the execution time of aligning two sequences from *O(mn)* down to *O(mn/p)* as long *as p=O(n/log n)*, where *m* and *n* are lengths of the two sequences to be aligned. Since *p* is limited by *O(n/log n)*, in the extreme, their algorithm has an execution time of *O(mlog n)*.

Besides parallel processing, heuristic methods are other commonly used approaches for speeding up sequence database searches. Heuristic methods are intended to gain computational performance, potentially at the cost of accuracy or precision. Popular alignment search tools in this category such as FASTA [29], BLAST [30] and BLAT [31] indeed gain some speed. However, the sensitivity is compromised. For sequence database searches, sensitivity refers to the ability to find all database sequences that resemble the query sequence above a threshold.

Here, we describe a parallel Smith-Waterman algorithm for general Single Instruction Multiple Data (SIMD) computers which achieves a significant improvement in speed over previous algorithms without sacrificing any sensitivity. This parallel algorithm requires an execution time of *O(m+n)* to align two sequences with lengths of *m* and *n*, respectively. Second, we propose a Very Large Scale Integration (VLSI) implementation of the parallel algorithm. Thirdly, we use the pipeline technique to overlap the execution times of alignment checking of database sequences. The resulting pipeline has a throughput rate of *O(1)* execution time per database sequence. Consequently, the time complexity of the proposed pipeline processor is *O(m+n+L)*, where *m* is the length of the query sequence and *n* and *L* are the maximum sequence length and the number of sequences in the database, respectively. Finally, we expand our pipelined architecture to perform all possible pair-wise alignments for a group of sequences. Checking all pairs of pair-wise alignments is essential to the overlap-layout-consensus (OLC) approach for de novo assembly [32-34]. Without any parallelization, $O(m^2L^2)$ time is required to align all possible pairs of sequences for a group of *L* sequences with a maximum sequence length of *m*. For the same task, our pipeline processor boasts a running time of *O(mL)*.

2. The Smith-Waterman Algorithm

The Smith-Waterman algorithm is used to compute the optimal local-alignment score. Let $A = a_1\, a_2\, \dots\, a_m$ and $B = b_1\, b_2\, \dots\, b_n$ be the two sequences to be aligned. A weight $w(a_i, b_j)$ is defined for every pair of residues a_i and b_j. Usually $w(a_i, b_j) <= 0$ if $a_i \neq b_j$, and $w(a_i, b_j) > 0$ if $a_i = b_j$. The penalties for starting a gap and continuing a gap are defined as g_{init} and g_{ext} respectively. The optimal local alignment score *S* can be computed by the following recursive relations:

$$E_{i,j} = \max \{ E_{i,j-1}-g_{ext}, H_{i,j-1}-g_{init} \} \quad (1)$$
$$F_{i,j} = \max \{ F_{i-1,j}-g_{ext}, H_{i-1,j}-g_{init} \} \quad (2)$$
$$H_{i,j} = \max \{0, E_{i,j}, F_{i,j}, H_{i-1,j-1}+w(a_i,b_j) \} \quad (3)$$
$$S = \max \{ H_{i,j} \}; \quad (4)$$

The values of $E_{i,j}$, $F_{i,j}$ and $H_{i,j}$ are 0 when i<1 and j<1. The Smith-Waterman algorithm is a dynamic programming algorithm requiring an $m\times n$ matrix, or so-called *alignment* matrix, to store and compute H, E, and F values column by column.

3. An Intra-sequence Parallelized Smith-Waterman Algorithm For SIMD Computers

Intra-sequence parallelization refers to parallelization within a single pair of sequences, in contrast to inter-sequence parallelization, where parallelization is carried out across multiple pairs of sequences.

Figure 1 shows the computational dependencies of the Smith-Waterman alignment matrix. This dependency structure illustrates the feature commonly exploited by existing parallelized Smith-Waterman algorithms in the literature. Our parallelization utilizes the following scheme: Since cells on a bottom-left to top-right diagonal have the same sum of indices, we number those diagonals with their sums of indices. Noticeably, cells of a diagonal do not depend on other cells of the same diagonal and thus can be computed simultaneously. Furthermore, cells of a diagonal only depend on cells of the previous two diagonals. Thus, the basic idea of the parallelization algorithm is to fill the matrix diagonal by diagonal starting from the top-left corner. Moreover, cells of a diagonal are filled by multiple processors simultaneously.

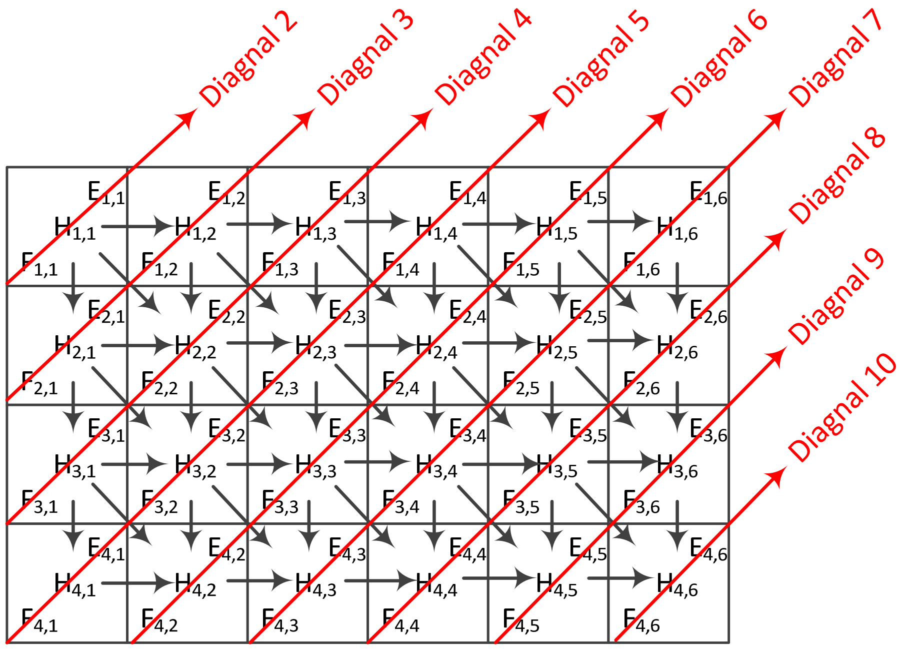

Fig. 1. Computational dependencies in the Smith-Waterman alignment matrix.

```
In parallel, all processor i, 1<= i <=m do
    E[i][0] = F[i][0] = H[i][0] = maxH[i] =  0;
In parallel, all processor i, 1<= i <=n do
    E[0][i] = F[0][i] = H[0][j] = 0;
for (diag=2; diag<=m+n; ++diag) {
    if (diag<=m+1) first = diag-1;
    else first = m;
    if (diag-n<=1) last = 1;
    else last = diag-n;
    In parallel, all processor i, first>= i >= last do  {
        $j_i$ = diag – i;
        E[i][$j_i$]  = max {E[i][$j_i$-1]-$g_{ext}$, H[i][$j_i$-1]-$g_{init}$};
        F[i][$j_i$]  = max {F[i-1][$j_i$]-$g_{ext}$, H[i-1][$j_i$]-$g_{init}$};
        H[i][$j_i$]  = max {0,E[i][$j_i$], F[i][$j_i$], H[i-1][$j_i$-1]+w($a_i$,$b_{ji}$)};
        if (H[i][$j_i$]>maxH[i]) maxH[i] = H[i][$j_i$];
    }
}
S = max { maxH[1], maxH[2], …max H[m]};
```

Fig. 2. The pseudo code.

The pseudo code of the parallel algorithm is given in **Figure 2**. Note that in the pseudo code, we use the in-parallel statement to indicate things to be done by processors simultaneously. The in-parallel statement has the following syntax:

```
In parallel, all processor i, lo<=i<=hi do {
        …
}
```

Only processors with processor numbers between lo and hi are activated. Also note that in the pseudo code, j_i is a variable for processor i only. In other words, different processors have different j's. Furthermore, array maxH is used for processors to keep track of the largest $H_{i,j}$. Since there are *m+n-1* diagonals, the loop repeats *m+n-1* times and thus the parallel algorithm has an execution time of *O(m+n)*.

4. A VLSI Implementation

First we design a small processing element according to the recursive relations (1), (2) and (3). Processing elements will be used to implement cells of Smith-Waterman's alignment matrix. Thus, each processing element will be numbered with the indices of its corresponding cell in Smith-Waterman's alignment matrix. As follows, processing element $PE_{i,j}$ will be used to compute values $E_{i,j}$, $F_{i,j}$ and $H_{i,j}$. As shown in **Figure 3**, processing element $PE_{i,j}$ consists of 3 registers $E_{i,j}$, $F_{i,j}$ and $H_{i,j}$ and several simple combinational circuits such as adders and comparators (for finding maximum of two or three values). Since $H_{i,j}$ depends on $E_{i,j}$ and $F_{i,j}$, processing element $PE_{i,j}$ has two computational states and thus requires two clock signals to complete its computation of values $E_{i,j}$, $F_{i,j}$ and $H_{i,j}$. Since the longest data path in the processing element consists of an adder and a comparator, the clock period, i.e. time between each clock signal, only needs to be set to the time delay caused by an adder and a comparator.

Each processing element $PE_{i,j}$ has 5 input ports and 3 output ports. Input ports A and B are for inputting residues a_i and b_j from the two sequences to be aligned. Input ports E_{in}, F_{in} and H_{in} are for inputting corresponding values from cells on which $PE_{i,j}$ depends. Output ports E_{out}, F_{out} and H_{out} are for exporting values to cells depending on $PE_{i,j}$. Additionally, in order to keep track of the largest $h_{i,j}$ value, register $H_{i,j}$ of processing element $PE_{i,j}$ is connected to register $maxH_i$ as shown in **Figures 3**.

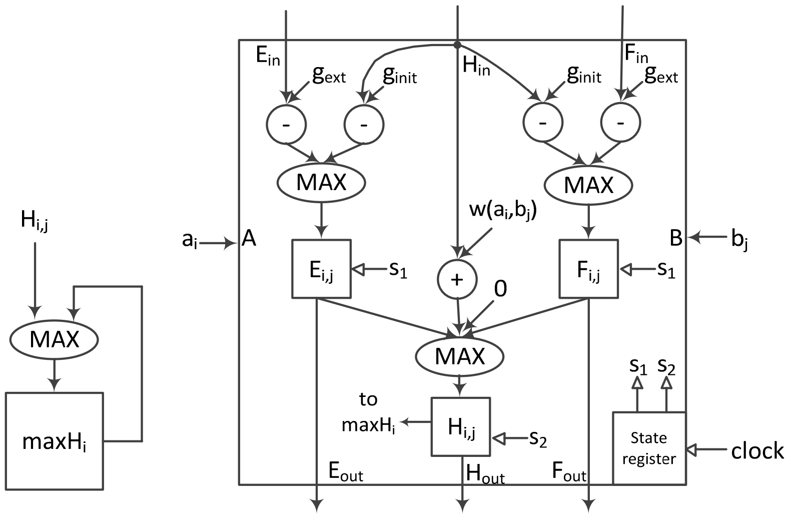

Fig. 3. Processing element $PE_{i,j}$ and register $maxH_i$.

Next, we map our algorithm into a VLSI circuit. **Figure 4** depicts our VLSI circuit at the register level for m=4 and n=6. Processing elements and registers $maxH_i$ are connected together according to recursive relations (1), (2), (3) and (4). As shown in **Figure 4**, processing elements are arranged into levels such that processing elements corresponding to cells of a diagonal are placed on the same level and thus will compute their values at the same time. For clarity of the logic of the VLSI circuit, levels are numbered with their corresponding diagonal numbers. Since the size of Smith-Waterman's alignment matrix is *mn*, the number of processing elements used in our VLSI circuit is *mn*.

5. A Pipeline Architecture for Sequence Database Searches

Since sequence database search is different from sequence alignment, first, we modify our processing element $PE_{i,j}$ as shown in **Figure 5**. Instead of keeping track of the largest $h_{i,j}$, the new processing element $PE_{i,j}$ will generate a SELECT signal when its $H_{i,j}$ value is above a threshold.

To speed up sequence database searches, a pipelined architecture is incorporated into the VLSI circuit. Pipelining is a natural concept for increasing the throughput of a system when processing a stream of data, even though pipelining cannot speed up the process of a single datum. The space-time diagram in **Figure 6** reveals the advantages of the pipelined architecture in processing database sequences. The diagram shows the succession of the levels in the pipeline with respect to time. From the diagram one can observe how independent sequences are processed concurrently in the pipeline.

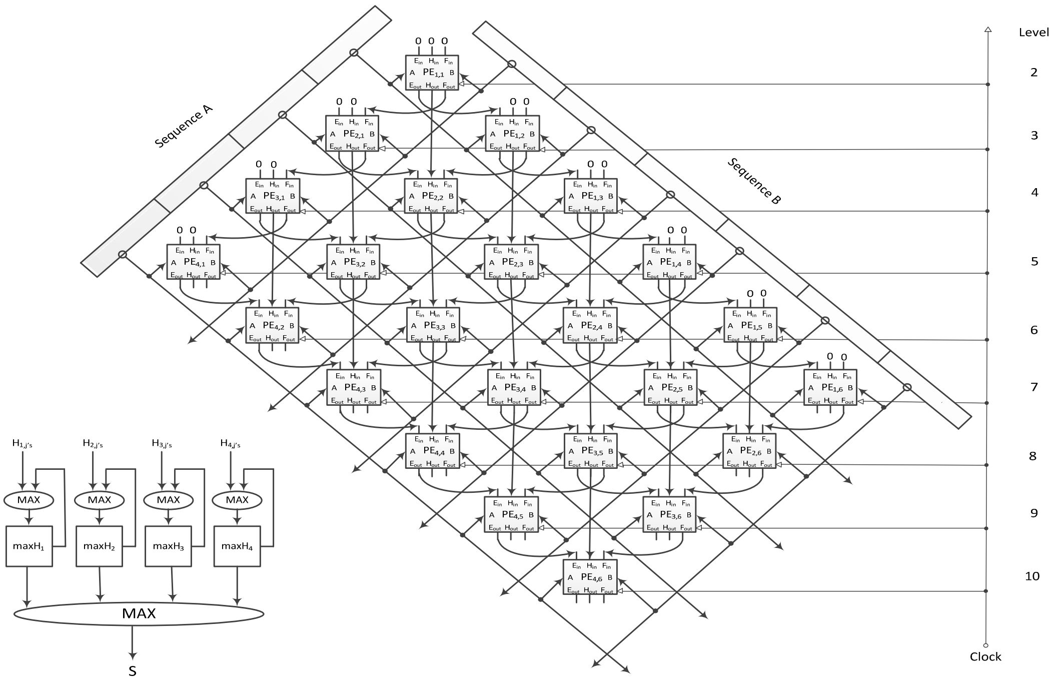

Fig. 4. The VLSI circuit, m=4, n=6.

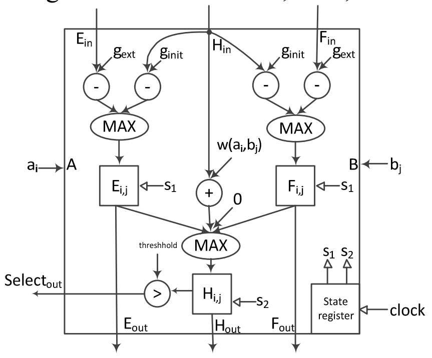

Fig. 5. Modified processing element $PE_{i,j}$.

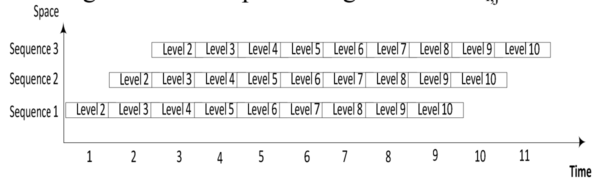

Fig. 6. Pipeline space-time diagram

With *m+n-1* levels in the VLSI circuit, we naturally employ a linear pipeline architecture with *m+n-1* stages (shown in detail in **Figure 7**). To do so, we add *m+n-1* registers to the VLSI circuit such that each register stores the database sequence of the corresponding stage (i.e. level). Additionally, each of these registers has an “S” flag which will be used to indicate whether the database sequence is selected or not. As shown in **Figure 7**, processing elements’ select signals are connected to S flags of the corresponding database sequence registers. Furthermore, according to recursive relation (3), $H_{i,j}$ depends on $H_{i-1,j-1}$ which is not in the immediate previous level of $H_{i,j}$. Thus, for the purposes of buffering and synchronization, $h_{i,j}$ buffer registers are added to the circuit to hold $H_{i,j}$ values. Since a processing element $PE_{i,j}$ needs two clock signals to compute its values, our pipeline takes 2 clock signals to shift a database sequence from one pipeline stage to the next.

After the first database sequence completes its alignment checking, an additional database sequence checking will complete with each subsequent stage time. Consequently, the pipeline processor has a throughput rate of one database sequence per two clock signals. In other words, the pipeline has a time complexity of *O(1)* time per database sequence. Moreover, because the first sequence requires *O(m+n)* time to complete its alignment checking, where *m* is the length of the query sequence and *n* is the maximum length of the sequences in the database, the total time complexity of the pipeline processor is *O(m+n+L)*, where *L* is the number of sequences in the database.

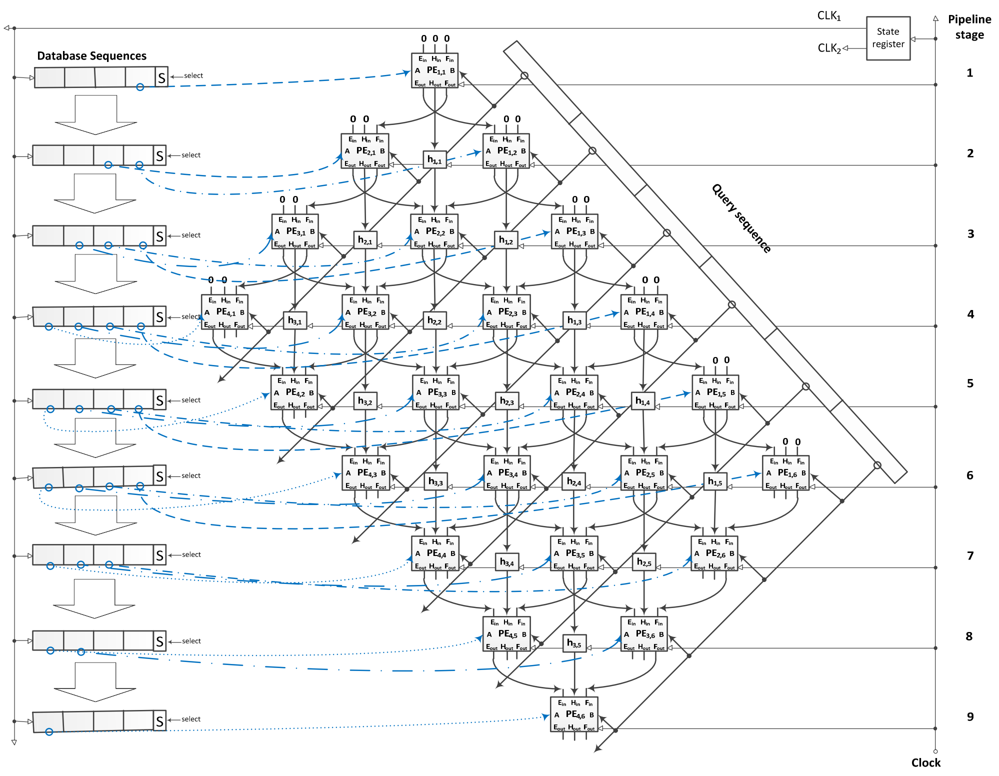

Fig. 7. The single-chip pipeline processor, m=4, n=6.

6. All pairs of pair-wise alignments

In bioinformatics, sequence assembly refers to aligning and merging fragments of a much longer DNA sequence in order to reconstruct the original sequence. Sequence assembly is needed as DNA sequencing technology cannot read whole genomes in one continuous string, but rather in small pieces of between 20 and 1000 bases, depending on the technology used. Typically, the short fragments, called reads, result from shotgun sequencing of genomic DNA [35, 36].

Sequence assembly can be categorized into two major types:

1. Comparative: assembling reads against an existing reference sequence, resulting in a sequence that is similar but not necessarily identical to the reference sequence. [37, 38]
2. *De novo*: assembling reads to create a full-length sequence in the absence of a reference genome sequence.[37-39]

The fundamental idea of comparative assembly utilizes Smith-Waterman's algorithm to map reads into locations in the reference sequence. Using a pipeline similar to that which we have described previously for sequence database searches, we can find locations of reads in the reference sequence simultaneously and thus vastly speed up sequence assembly.

In contrast, *de novo* assembly offers more challenges. *De novo* assembly algorithms were first developed in the 1980's and based on the overlap-layout-consensus (OLC) approach [32-34]. An essential and extremely time-consuming step of the OLC approach involves constructing a so-called overlap graph from reads. Constructing the graph requires performing a pair-wise alignment for each and every pair of reads and thus $O(m^2L^2)$ time, where L is the number of reads and m is the maximum length of reads. Replacing sequential alignments with our proposed pipeline architecture reduces the execution time significantly to $O(mL)$.

6.1 A pipeline architecture for all pairs of pair-wise alignments

Figure 8 is a schematic diagram of our pipelined architecture for performing all pairs of pair-wise alignments. Here, reads are shifted across the pipeline and compared with each other. As used in our database search pipeline, there are stage registers and a fixed register in the pipeline. At every step, reads in each of the stage registers will be simultaneously compared with fixed register reads. Each stage register holds a single read at a time while the fixed register will initially hold a single read and accumulate an additional read with each step.

Each read will initially be inserted into the first stage register and passed along to the next stage register with each time step. After a read has shifted through all stage registers, i.e. completed alignment with reads in the fixed register, it will be added to the fixed register. Thus, all subsequent reads traveling through the stage registers will be compared with this newly added fixed register read. When the last read has shifted through all stage registers, all pairs of pair-wise alignment are complete.

Let L be the number of reads and m is the maximum length of reads. Since only $L-1$ reads need to be in the fixed register, the size of the fixed register is no greater than $m(L-1)$. Since the length of the pipeline is the size of the fixed register, the processing time of the pipeline is $O(mL)$.

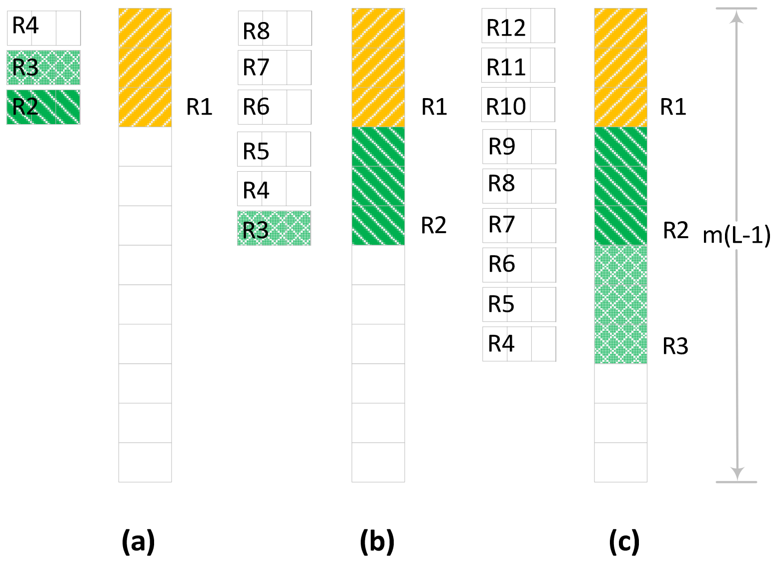

Fig. 8. Pipeline architecture for performing all pairs of pair-wise alignments.

6.2 A circular pipeline architecture

Given that there are L reads, at any instant, at most L stages of the pipeline are actually used. In order to save on hardware cost, we can arrange our pipeline as a ring with a size of L and shift reads around the ring (**Figure 9**). To complete all pairs of pair-wise alignments, reads need to go around the ring about m times. The total processing time remains the same. However, the number of processing elements of the pipeline is reduced from m^2L to mL and the number of shift registers is reduced from mL to L.

7. Parallel Efficiency

Parallel efficiency is an important performance metric for parallel processing. Parallel efficiency, E, defined as previously by others [40] is $E = T_{seq}/(p \times T_{par})$, where T_{par} is the runtime of the parallel algorithm, T_{seq} is the runtime of the best sequential algorithm and p is the number of processors used. This value, typically between zero and one, quantifies how well processors are utilized in solving the problem. An efficiency of 1 is ideal, indicating that 100% of the time processors are used. **Table 1** summarizes the efficiencies of our parallel algorithm and pipelines. Clearly, our pipelines use processing elements efficiently. Note that, to be exact, parallel efficiency E for our sequence database search pipeline is $O(mnL)/O(mn \times (m+n+L))$ which is $O(L/(m+n+L))$. However, in reality, L is much bigger than $m+n$. Thus, E for our sequence database search pipeline is $O(1)$.

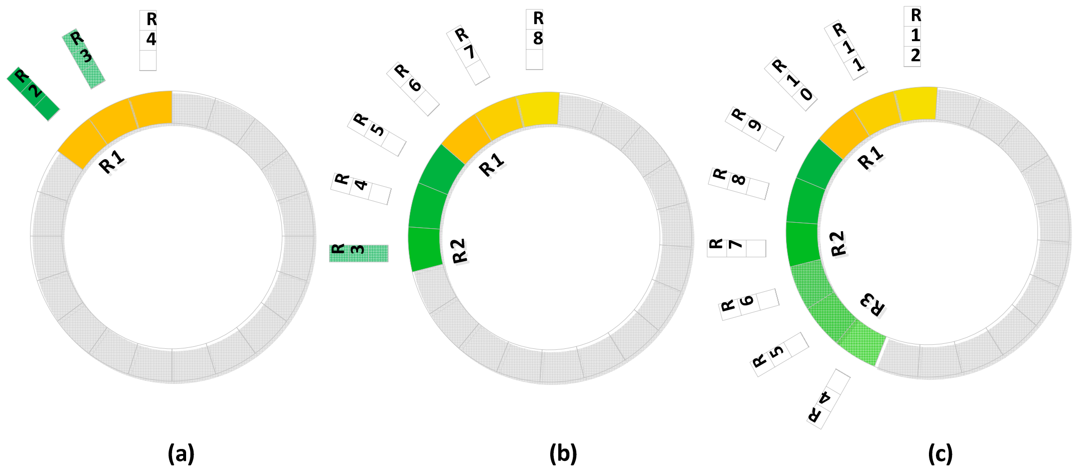

Fig.9. A circular pipeline architecture.

Table 1. Algorithm Runtimes and Corresponding Parallel Efficiencies.

	VLSI circuit for parallelized Smith-Waterman Algorithm	Pipeline for sequence database search	Circular pipeline for all pairs of pair-wise alignment
P:Number of PE's used	*mn*	*mn*	*mL*
T_{seq}	$O(mn)$	$O(mnL)$	$O(m^2L^2)$
T_{par}	$O(m+n)$	$O(m+n+L)$	$O(mL)$
$E= T_{seq}/(p\times T_{par})$	$O(1/(m+n))$	$O(1)$	$O(1)$

8. Practical Considerations

For *de novo* assembly, *L* is in billions and *m* is in hundreds. Since our circular pipeline requires *O(mL)* processing elements, our circular pipeline needs an enormous hundred billions of processing elements. However, as of today's technology, 100 trillion transistors could fit on a single chip [41]. Since a register or a simple combinational circuit such as adder or comparator doesn't need thousands of transistors, 100 trillion of transistors should be enough for hundred billions of processing elements. As a result, it's possible to implement our pipeline processor in a single VLSI microchip.

9. Conclusion

In this paper, we have described an *O(m+n)* time intra-sequence parallelized Smith-Water algorithm for general SIMD computers, where *m* and *n* are lengths of the two sequences to be aligned. We have shown a VLSI implementation of the parallel algorithm. We have illustrated that by incorporating a pipelined architecture into the VLSI circuit, we can speed up sequence database searches without sacrificing sensitivity. The resulting pipeline processor can do sequence database searches at the speed of *O(m+n+L)*, where *m* is the length of the query sequence and *n* and *L* are the maximum sequence length and the number of the sequences in the database, respectively. Moreover, we have demonstrated that our pipelined processor can do all pairs of pair-wise alignments in *O(mL)* time for a group of *L* sequences with a maximum sequence length of *m*. Our novel architecture may greatly change the course of the sequencing industry, particularly in next-generation technology development, and accelerate the pace of biological analyses and discoveries.

10. References

[1] TF Smith, MS Waterman, *Identification of common molecular subsequences.* J Mol Biol. 147, 195–197 (1981).

[2] O. Gotoh, *An improved algorithm for matching biological sequences.* J Mol Biol. 162, 705–708 (1982).

[3] D. Brutlag, J. Dautricourt, R. Diaz, J. Fier, B. Moxon, and R. Stamm, *Blaze - an implementation of the Smith-Waterman Sequence Comparison Algorithm on a Massively Parallel Computer*, Comput. Chem., vol. 17, pp. 203-207, 1993.

[4] M Borah, RS Bajwa, S Hannenhalli, MJ Irwin, *A SIMD solution to the sequence comparison problem on the MGAP.* Application Specific Array Processors, pp336-45, Los Alamitos, CA: IEEE CS, 1994.

[5] B Alpern, L Carter, KS Gatlin, *Microparallelism and high performance protein matching.* Proceedings

of the 1995 ACM/IEEE Supercomputing Conference, San Diego, California, Dec 3-8, 1995.

[6] R Hughey, *Parallel hardware for sequence comparison and alignment.* Comp Appl Biosci. 1996;12:473–479.

[7] A Wozniak, *Using video-oriented instructions to speed up sequence comparison.* Comput Appl Biosci. 13, 145–150 (1997)

[8] X. Huang, *A Space-Efficient Parallel Sequence Comparison Algorithm for a Message-Passing Multiprocessor*, Int'l J. Parallel Programming, vol. 18, no. 3, pp. 223-239, 1989.

[9] T Rognes, E Seeberg, *Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors.* Bioinformatics. 16, 699–706 (2000).

[10] Y. Yoshiki, M. Yosuke, M. Tsutomu, and K. Akihiko, *High Speed Homology Search Using Run-Time Reconfiguration*, Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications: Springer-Verlag, 2002.

[11] C. Janaki and R. Joshi, *Accelerating comparative genomics using parallel computing*, Silico Biol., vol. 3, pp. 429-440, 2003.

[12] S. Rajko and S. Aluru, *Space and Time Optimal Parallel Sequence Alignments, IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 11, pp. 1070-1081, Nov. 2004.

[13] T. Oliver, B. Schmidt, D. Nathan, R. Clemens, D. Maskell: *Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW.* Bioinformatics 21(16): 3431-3432 (2005).

[14] ITS Li, W Shum, K Truong, *160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA).* BMC Bioinformatics. 8, 185 (2007).

[15] M Farrar, *Striped Smith-Waterman speeds database searches six times over other SIMD implementations.* Bioinformatics. 23, 156–161 (2007).

[16] Jiang at al: *A Reconfigurable Accelerator for Smith–Waterman Algorithm,* IEEE TCAS-II, 54(12), pp. 1077-1081, 2007

[17] Z. Nawaz, M. Shabbir, Z. Al-Ars, and K. Bertels, *Acceleration of smith-waterman using recursive variable expansion,* in DSD-2008, pp. 915–922, September 2008.

[18] A Szalkowski, C Ledergerber, P Krähenbühl, C Dessimoz, *SWPS3 - fast multithreaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2.* BMC Res Notes. 1, 107 (2008).

[19] A Wirawan, CK Kwoh, NT Hieu, B Schmidt, *CBESW: Sequence Alignment on the Playstation 3.* BMC Bioinformatics. 9, 377 (2008).

[20] Y. Munekawa, F. Ino, and K. Hagihara, *Design and implementation of the Smith-Waterman algorithm on the CUDA-compatible GPU*, presented at BioInformatics and BioEngineering, 2008. BIBE 2008. 8th IEEE International Conference on, 2008.

[21] W Rudnicki, A Jankowski, A Modzelewski, A Piotrowski, A Zadrożny, *The new SIMD Implementation of the Smith-Waterman Algorithm on Cell Microprocessor.* Fund Infor. 96, 181–194 (2009)

[22] Y Liu, DL Maskell, B Schmidt, *CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units.* BMC Res Notes. 2, 73 (2009).

[23] Ł Ligowski, WR Rudnicki, *An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases*. Eighth IEEE International Workshop on High Performance Computational Biology, Rome, Italy, 2009.

[24] K. Benkrid, Y. Liu, and A. Benkrid: *A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment.* IEEE Trans. VLSI Syst. 17(4): 561-570 (2009)

[25] Y Liu, B Schmidt, DL Maskell, *CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA_enabled GPUs based on SIMT and virtualized SIMD abstractions.* BMC Res Notes. 3, 93 (2010).

[26] Ł Ligowski, WR Rudnicki, Y Liu, B Schmidt, *Accurate Scanning of Sequence Databases with the Smith-Waterman Alg.* GPU Comput. Gems, Emerald Edition. (Morgan Kaufmann, 2011), pp. 155–157.

[27] R Torbjorn, *Faster Smith-Waterman database searches with inter-sequence SIMD parallelization*. BMC Bioinformatics 12:221 2011.

[28] Y. Yamaguchi, et al., *FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design Reconfigurable Computing: Architectures, Tools and Applications*, vol. 6578, Lecture Notes in Computer Science: Springer Berlin / Heidelberg, 2011, pp. 181-192.

[29] WR. Pearson, DJ. Lipman, *Improved tools for biological sequence comparison.* PNAS, 85 (8): 2444-8. (1988)

[30] SF. Altschul, W. Gish, W. Miller, EW. Myers, DJ. Lipman, *Basic local alignment search tool.* J Mol Biol 215 (3): 403–410. (October 1990)

[31] WJ. Kent, *BLAT--the BLAST-like alignment tool.* Genome research 12 (4): 656–664. (2002)

[32] E. W. Myers, G. G. Sutton, A. L. Delcher, et.al., *A whole-genome assembly of Drosophila.* Science (New York, N.Y.), vol. 287, no. 5461, pp. 2196–2204, Mar. 2000.

[33] Batzoglou, S. Jaffe, D. Stanley, K. Butler, J. Gnerre, S. Mauceli, E. Berger, B. Mesirov, J. and Lander, E. *Arachne: A whole-genome shotgun assembler,* Genome Res. 2002 Jan;12(1):177-89.

[34] Margulies, M. Egholm, W. E. Altman, S. Attiya, et.al. *Genome sequencing in microfabricated high-density picolitre reactors*, Nature, no. 7057, pp. 376–380, Sep.2005

[35] Anderson, S (1981). Shotgun DNA sequencing using cloned DNase I-generated fragments. Nucleic Acids Research **9** (13): 3015–27, 1981.

[36] J. R. Miller, S. Koren, and G. *Sutton, Assembly algorithms for next generation sequencing data,* Genomics, vol. 95, no. 6, pp. 315–327, Jun.2010.

[37] S. Bao, R. Jiang, W. Kwan, X. Ma, and Y.-Q. Song*, Evaluation of next-generation sequencing software in mapping and assembly*, J Hum Genet, vol. 56, no. 6, pp. 406–414, Jun. 2011.

[38] Applied Biosystems, *Applied Biosystems SOLiD 3 Plus System: De Novo Assembly Protocol, Life Technologies Corporation,* Tech. Rep.,2010

[39] L. Jorde, Encyclopedia of Genetics, Genomics, Proteomics, and Bioinformatics: 8 Volume Set. John Wiley & Sons Ltd., 2005.

[40] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors, Volume 1,* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[41] Q. Yuan, H. Hu, J. Gao, F. Ding, Z. Liu, and B. Yakobson, *Upright Standing Graphene Formation on Substrates, J. Am. Chem. Soc.*, **2011**, *133* (40), pp 16072–16079.